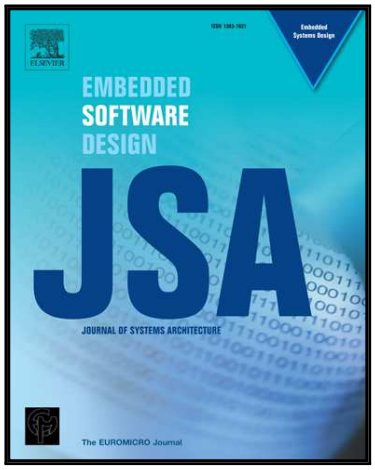
Article:
Dai, Xiaotian orcid.org/0000-0002-6669-5234 and Burns, Alan orcid.org/0000-0001-5621-8816 (2020) Period Adaptation of Real-Time Control Tasks with Fixed-Priority Scheduling in Cyber-Physical Systems. Journal of systems architecture. 101691. ISSN 1383-7621

https://doi.org/10.1016/j.sysarc.2019.101691



Period Adaptation of Real-Time Control Tasks with Fixed-Priority Scheduling in Cyber-Physical Systems

Xiaotian Dai, Alan Burns

PII: S1383-7621(19)30498-9
DOI: https://doi.org/10.1016/j.sysarc.2019.101691
Reference: SYSARC 101691

To appear in: *Journal of Systems Architecture*

Received date: 26 June 2019
Revised date: 21 October 2019
Accepted date: 23 November 2019

Please cite this article as: Xiaotian Dai, Alan Burns, Period Adaptation of Real-Time Control Tasks with Fixed-Priority Scheduling in Cyber-Physical Systems, *Journal of Systems Architecture* (2019), doi: https://doi.org/10.1016/j.sysarc.2019.101691

Period Adaptation of Real-Time Control Tasks with Fixed-Priority Scheduling in Cyber-Physical Systems

Xiaotian Dai[a], Alan Burns[a]

[a]*Department of Computer Science, University of York, United Kingdom*

Abstract

Long-lived, non-stop cyber-physical systems (CPS) are subject to evolutionary changes that can undermine the guarantees of schedulability that were verified at the time of deployment. At the same time, knowledge gleamed from extended periods of execution ~~will~~ can be exploited to reduce the uncertainties that were inevitably presented in the system models that are used to define the temporal behaviours of the control tasks. In this paper we utilise this knowledge and present an adaptation method that actively extends the period of control tasks at run-time based on historical measurements. This can lead to lower power consumption or to the accommodation of increased computation resource demands from other components of the CPS. The method relies on online monitoring and model-based prediction to degrade control performance while having a minimal and acceptable impact on ongoing operations. Cloud-based computing is used to facilitate decision making and offload the local computation. We evaluate the effectiveness of the proposed method through ~~experiments of~~ control-scheduling co-simulation.

Keywords: Cyber-Physical Systems, Control-Scheduling Co-Design, Period Adaptation, Adaptive Scheduling, Digital Twin, Model-Based Engineering, Ada

1. Introduction

Cyber-physical systems (CPS) often contain a number of control functions that require long-lived and non-stop execution. During the operation of a CPS, considerable knowledge about its execution behaviour can be obtained and exploited. For example, if long-term trends in resource usage are identified [1], adaptation can be made to accommodate additional computation requirements. In this paper, an adaptation method that uses online monitoring and model prediction is introduced, which can be used to reduce the resource requirements of the control tasks in a CPS.

~~Before a system is deployed~~ Prior to the deployment of a system in the field, there is only limited knowledge about the system's uncertainties and the interactions between different aspects of the external environment etc. Also the uncertainties introduced by scheduling

Email addresses: xiaotian.dai@york.ac.uk (Xiaotian Dai), alan.burns@york.ac.uk (Alan Burns)

are not able to be fully accounted for during the design phase, and the worst-case execution time is not always accessible or precise. This means that the system model is inevitably conservative. To improve the situation, the real behaviour of a system can be observed, and the control performance can be assessed at run-time. If there is evidence that control performance can be safely degraded, system schedulability can be improved by increasing the control intervals of control-related tasks. This released capacity can then be used to:

- cater for non-control tasks in the system that are experiencing evolutionary increases in demand [2], e.g. tasks concerned with communication, data processing, system monitoring, diagnosis, fault recovery, decision making etc.;
- save energy in ~~battery powered~~ systems by reducing processor speed, core usage, activation time, etc [3]. Most modern processors support dynamic power management [4], e.g., power saving mode, dynamic voltage and frequency scaling (DVFS), etc. For example, the reduced equivalent CPU utilization will create idle time during which the CPU can stay in the sleep mode. Energy saving is particular beneficial to battery-powered systems;
- improve the quality of other aspects of the system which have been implemented using methods that can benefit from longer execution times, for example, anytime algorithms [5, 6, 7];
- accommodate increased resource demand due to future system upgrades, and/or increase the resilience of the system due to task overruns.

In this paper, we present an online adaptation method that uses ideas of ~~digital twin~~ digital twining and model-based engineering, which is based on feedback measurements and a decision planner, to safely degrade control performance in a predictable and managed way. The desired control quality is defined statistically through a degradation degree parameter. This parameter reflects the tolerance of degraded control performance that is acceptable. Depending on the application, the acceptance region can be relative large or can be very limited. In both cases, we assume one or more control tasks are initially running with conservative periods which are then gradually increased in small and controlled minor steps. During this adaptation process, performance predictions are made in advance, and a step change in period is only made if sanctioned by the predication. The consequence of the period change is monitored and feedback is used to improve the predictor.

This paper is organised as follows. Background of real-time control is reviewed in Section 2. A general overview of the adaptation method is given in Section 3. The performance prediction and the run-time system support is discussed in Section 4 and Section 5, respectively. In Section 6 an evaluation based on a control-scheduling co-simulator is made to demonstrate and ~~verify~~ evaluate the effectiveness of the proposed method, followed by a discussion on some ~~aspects~~ implementation details of the approach in Section 7. Finally, a summary of the work is given in Section 9.

2. Real-Time Digital Controller Implementation

A real-time control task is the entity that executes the software implementation of a digital feedback controller. A general structure of feedback control with a digital controller in the loop is given in Fig. 1. To implement such a controller on an embedded platform, the control functions need to be abstracted into individual tasks~~. After a digital control task is running on a computer, the controller will behave differently than the ideal periodic execution according to the applied scheduling algorithm.~~ and be scheduled with a real-time scheduler, which will introduce interference and constraints on the control period.

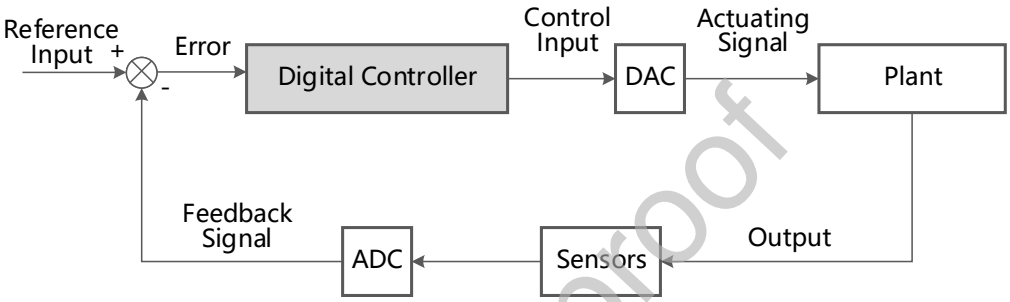


Figure 1: A digital controller in a feedback control system

2.1. Control Loop Timing

In the control community, it is often assumed that sampling and control are performed equidistantly and simultaneously with a fixed interval. However, as a digital controller is running on a computer, the controller will behave differently than the ideal periodic execution according to the applied scheduling algorithm.

Consider a system with three control tasks that are scheduled by a fixed priority scheduler (FPS), in which tasks are scheduled preemptively according to their priorities (a smaller task index indicates a higher priority). A timing diagram of this case is shown in Fig. 2. Due to the nature of sharing resources in a multiprogramming environment, the timing of a control task is not fully deterministic. Among all three tasks, task 1 has the highest priority and hence is not suffering from interferences. However having the lowest priority, task 3 has the largest interferences and jitter. This example shows that in a multiprogramming environment, a control task without the highest priority will be occasionally preempted and suffer interferences from higher priority tasks in the same system. This process will introduce artefacts such as sampling jitter and control delay which will affect control outcomes [8, 9].

To explain the details more, an illustration of the timing of a single control task is given in Figure 3. In the diagram, h_i is the task period, $\tau_{s,j}$ is jth sampling delay and $\tau_{\mathtt{io},j}$ is jth input-output latency. The sampling delay and jitter can be eliminated by using a programmable ADC that is synchronised to the task period. However, the scheduling-introduced input-output latency, or control delay, cannot be precisely predicted as it could be different for each job instance. From the scheduling point of view, if $\tau_{s,j} = 0$, this is

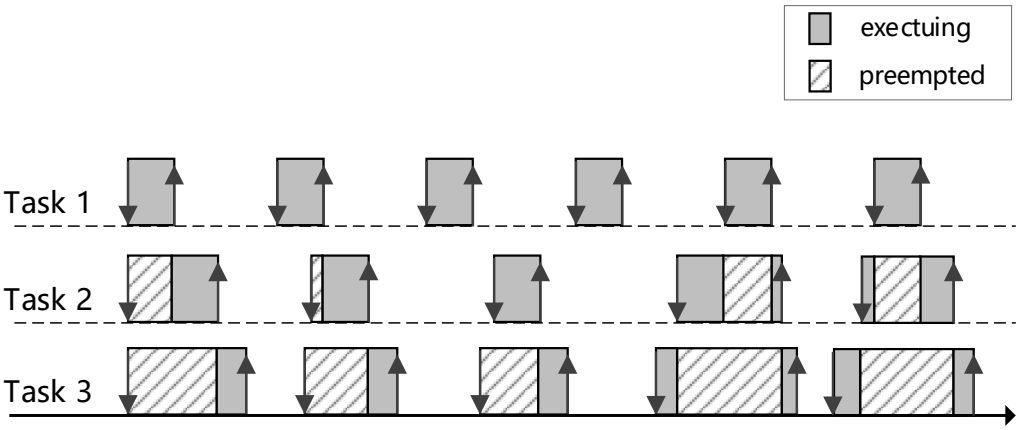


Figure 2: Task timing of multiple tasks scheduled by FPS

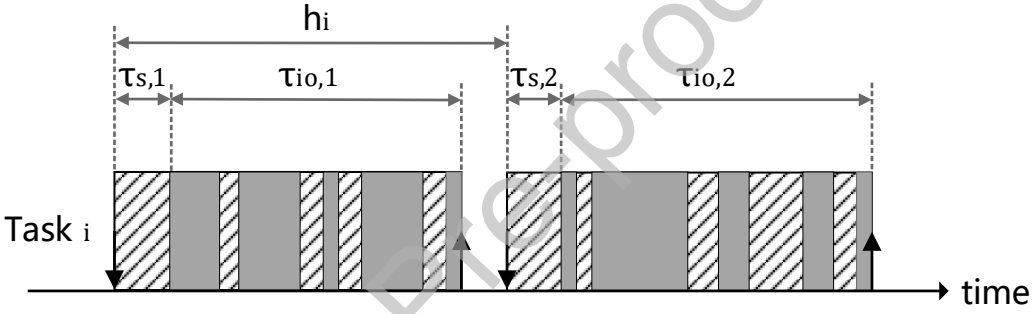


Figure 3: Control task timing (single task)

equivalent to the control task's response time, which consists of its own execution time, the interference time from higher priority tasks and the blocking time from lower priority tasks.

The influence of timing variations is dependent on the controlled plant and the controller. Some control systems are robust towards variations in control interval and latency, but others are less tolerant. A study on the influence of jitter on control performance is described in [10], where the notion of 'jitter margin' is introduced. Overall, it is hard to analytically work out the exact effects on control that are introduced by task scheduling.

2.2. Control Performance Evaluation

To evaluate the goodness of control from run-time measurements, a comprehensive performance index (PI) should be used that can quantitatively evaluate the control quality. It is important, as a first step of our method, that the performance metrics are defined numerically and generally.

~~To analyse the performance of a real-time controller, some numerical performance metrics need to be defined.~~ There are many commonly used criteria for evaluating the performance of a designed controller, for example, percentage of overshoot, settling time, deviation from a reference output, control error variance, etc [11]. A Performance Indicator (PI) can be

seen as a numeric evaluation of the performance of the target control system. To meet the requirement of this work, it should be able to reflect the true quality of a controller running under different conditions. Ideally, it should satisfy the following properties:

- it can be quantified and is numerically comparable;
- it can be normalised;
- it can reflect relative long-term behaviour;
- it is insensitive to initial states and noise.

Control error, i.e., the difference between the desired and the actual output, is a straightforward measure of the instant performance. However, this metric has large fluctuations as each of its evaluations is dependent on the current system state. ~~To smooth short-term variations, some form of integral error is applied to evaluate controllers, e.g., integral of absolute error (IAE), integral of time-weighted absolute error (ITAE), integral of squared error (ISE), etc. In some circumstances this integral error is known as control cost.~~ To handle this issue, we use the integral of absolute error (IAE) to smooth short-term variations, which is a PI that is often used in optimal controller designs.

Note that a higher ~~control cost~~ IAE ~~means~~ indicates worse performance. When the cost is measured at run-time, there could still be inconsistent variations due to scheduling and system noise (even it is smoothed by integration). Thus a ~~range~~ population of possible costs could be observed, that can be ~~represented~~ modelled as a distribution.

2.3. Control Task Period

The control performance, or Quality-of-Control (QoC), of a digital control system is largely affected by the sampling period of the controller task. In the work of [12], it is shown that the performance of a digital controller has a monotonic decreasing relationship in regards to the control period. This claim is generally true for most systems, although a counterexample is given in [13] for a non-inverted pendulum.

The selection of a control task's period depends on the dynamics of the controlled plant, the desired control performance, and the resources that are available on the hardware platform. It is often hard for a control engineer to determine which is the right period to use at the system design stage. Following a rule-of-thumb or experimental simulations is good common practice. However as no scheduling effects are considered, the selected period could be too pessimistic and thus waste system resources. Ideally, a period should be adequate to satisfy control performance under required specifications, whilst using the least CPU resources. In the following sections, we will discuss an adaptation method that attempts to deliver this behaviour after deployment.

3. Adaptation Method Overview

This work is identified as control-scheduling co-design [12], in which scheduling efforts are considered explicitly in the design process of real-time controllers. Resource constraints are also considered during the design of a digital controller.

Our work is also a form of feedback scheduling [13, 14], in which the scheduler has the ability to monitor system states, and make corresponding actions by adjusting scheduling parameters (e.g., task attributes such as task periods, execution times, deadlines and priorities). In this work, we only focus on changing task periods.

In contrast to off-line optimal control period assignment [10, 11], our method is an online adaptation method. The philosophy applied is to tune control task periods gradually and slowly, in which changes are made across a large time span, e.g., hourly or daily. This makes our approach much less dynamic than other feedback-based period allocation methods, in which a decision of change is made every tens or hundreds of milliseconds. This work is also related to graceful degradation [15], in which planned and pre-designed degradation is made in order to avoid serious system failures.

As in each adaptation cycle only a small change is applied and the consequence of the change is also observed and considered, our method is less aggressive than some of the existing adaptation methods, for example, state-aware and resource-aware feedback [16, 17, 18].

3.1. System Structure

In this system, it is assumed that each element of the physical plant is controlled by an individual control task executing on an embedded computer, which has connectivity to a more powerful machine 'in the cloud'. All tasks are executing concurrently and independently. Each task is responsible for sampling, updating system state and calculating control signals.

Fig. 4 shows the basic structure of the proposed method. The system is composed of a server and one or more clients. The client/server structure distributes the computational load that is required, as the server has much more processing power than the local embedded computer. The traces of control and scheduling performance are measured at the local system with a monitor module, and transferred to the cloud server for processing and analysis. The planner on the cloud will make a decision if a longer period can be applied, utilising a model-based predictor and the run-time observations. The observed data will also be used to update the prediction model which forms a feedback loop.

To use this method, some general assumptions on the properties of the deployed system are applied:

1. The embedded computer (local system) consists of a uniprocessor and a preemptive scheduler using fixed-priority scheduling (FPS);

2. All tasks in the system are released periodically and are initially schedulable, given the control tasks are using periods that can satisfy control specifications. The system has the ability to monitor task execution and response times, which is often supported

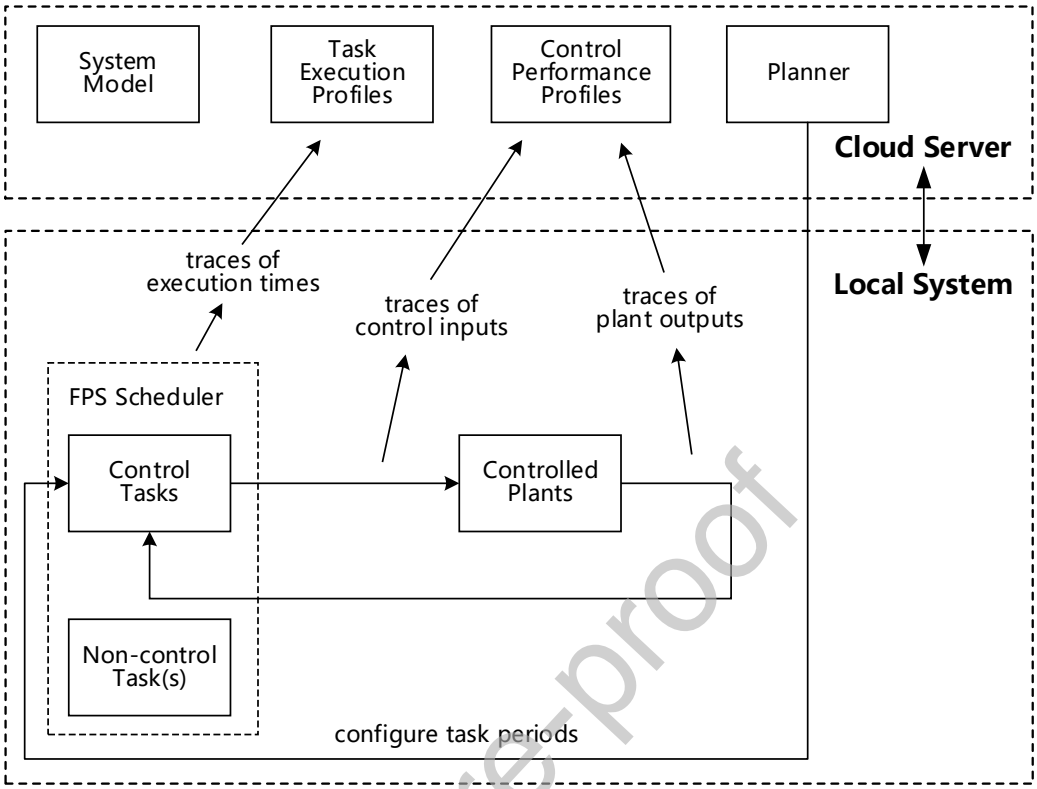


Figure 4: Block diagram of the proposed adaptation method

by POSIX-compliant kernels and real-time programming languages such as Ada (see Section 7.3);

3. The system itself has limited resources but has connectivity to a more powerful cloud server, e.g., IBM Bluemix, Amazon Web Services (AWS) or Google Cloud. The connection link does not need to be reliable or timing predictable.

3.2. Task Model and Problem Formulation

Given a control application that is represented as a ~~task set~~ taskset $\Gamma = \{\Gamma_c \cup \Gamma_{nc}\}$, in which Γ_c is the subset of all the control tasks and Γ_{nc} represents the subset of tasks that are not control-related. For each control task $\tau_i \in \Gamma_c$, a flexible task model is used [14], of which the task period is a variable parameter. A task is defined, using the normal symbols as $\tau_i \equiv \{C_i, T_i^0, T_i, D_i\}$. T_i^0 is the initial period as well as the lower boundary on the period, and T_i is the current period. The schedulability of the initial ~~task set~~ taskset is checked through response time analysis by using optimal priority assignment based on the initial periods.

The control aspect of a system can be represented as time-domain differential equations that describe the relationship between the control signal inputs and the system response. Define the controlled plant of control task τ_i as P_i. The system dynamic model of P_i is

represented in the standard state-space form:

$$\begin{cases} \dot{x}_i(t) = A_i x_i(t) + B_i u_i(t) + \omega_i(t) \\ y_i(t) = F_i x_i(t) + e_i(t) \end{cases} \tag{1}$$

in which $x_i(t)$ is the system states vector; $\dot{x}_i(t)$ is the first derivative of x_i; $u_i(t)$ is the control input; $y_i(t)$ is the system output; A_i is the system dynamic matrix; B_i is the input matrix; F_i is the output matrix; $\omega_i \sim \mathcal{N}(0, \sigma_\omega^2)$ is system process noise, and $e_i \sim \mathcal{N}(0, \sigma_e^2)$ is measurement noise. The notation $\mathcal{N}(m, n)$ means normal distribution with mean m and variance n. This model is used in the controller design as well as the prediction module in the cloud.

~~The optimization objective~~ The problem of adaptation can be understood as an optimization problem, in which the optimization objective is to minimise the overall resources used by the control tasks under given control quality constraints, which is formulated as follows:

$$\begin{aligned} \underset{T_i}{\text{minimise}} \quad & \sum U_i = \frac{C_i}{T_i}, i \in \Gamma_c \\ \text{subject to} \quad & \frac{PI_i(T_i)}{PI_i^0} \geq 1 - \lambda_d(i), i \in \Gamma_c \end{aligned} \tag{2}$$

The Performance Index (PI) is defined as an inverse of a control cost J (see Section 4.2), i.e., $\text{PI} = 1/J$. Using PI, the performance of period T_i and an arbitrary period T_i' can be compared. The degradation factor λ_d is defined as the fraction of the expected performance at a desired period, $\text{PI}_i(T_i)$, and the ideal expected performance PI_i^0 (when $T_i = T_i^0$). This introduced design parameter is used to make tradeoffs between task utilization and control performance. It is important that the performance index should be comprehensive and should also be a monotonic decreasing function with regard to the task period. By defining the degradation factor, the system designer can control the tolerance of ~~QoC (Quality of Control)~~ quality-of-control (QoC) degradation as a consequence of manipulating periods. The subscript i in ~~Equation~~ Eq. (1) and ~~Equation~~ Eq. (2) will be omitted if there is only one control task.

4. Performance Prediction

It is important for the system to determine the consequence of applying a new period, and making advance predictions is a straight-forward way ~~of estimating such influence~~. The adaptation relies on a performance prediction process, which is done through a model-based performance predictor using a Monte Carlo model that runs in the cloud server. The predictor has the ability to predict the performance distribution of a given digital controller when operating at a particular rate with a given task model. Monte Carlo is a method for evaluating a model that is complex, non-linear, or involves more than just a couple of uncertain parameters. Monte Carlo approximates results (i.e., the predictions) from a large number of repeated experiments through random sampling.

In this work, a Monte Carlo method is used for analysing how task scheduling uncertainties and variations would propagate to affect control system performance. It is used as the original control problem is hard to solve by a deterministic and analytical calculation. From a hybrid system point of view, each control action introduces a jump - i.e., a sudden change in system dynamics. Although control jobs are released periodically, the actual execution and outputs are not equally distributed in time. The overall control output is therefore the consequence of the contributions of multiple control job releases.

4.1. Monte Carlo Predictor

The overall predictor structure is shown in Fig. 5. The predictor is formed of a simulator, a system dynamic model, a taskset model and a correction model to generate performance profiles. The Monte Carlo simulator module is a hybrid system that is formed of a discrete and a continuous computation models. The discrete model (Fig. 6) is a timed finite state machine. In the diagram, t_1: is the delay due to phasing of the first released job after the operation point is changed, $t_1 \in (0, T_i)$. The worst-case is when the operational point changes right after the task is released. In this case, the control task will only be aware of the change after its next release; t_2: is the execution delay after the task is released due to interference from higher priority tasks; t_3: is the input-output latency, which is partly due to task interference and partly from task execution; t_4: the delay for the next release of the job; *cond*: is a conditional to terminate the simulation. This termination criterion is either the system has reached steady state or the maximum allowed simulation time has passed.

The continuous module simulates control system dynamics with an ODE solver. It takes inputs of control signals, and produces system responses as output, which can then be used to calculate the performance index. The system dynamic model that is used can either be obtained from first principles or from empirical modelling. If the system model is in the form of a transfer function, it will be converted into a state-space model.

During the process of the simulation, the continuous model receives control inputs and sends plant outputs to the discrete model, while the discrete model decides when and how to update the input signal, according to the states of the discrete timing model. Collaboratively, these two subsystems simulate the real run-time behaviour of the digital controller.

4.2. Predicting Control Performance

The performance of a designed controller is quantitatively evaluated by control cost J. In our work, the Integral of Absolute Error (IAE) is used to describe the cost, which has the general form:

$$J = \int_0^{tss} |e(t)| \cdot dt \tag{3}$$

The control error, $e(t)$, is defined as the difference between the desired reference $r(t)$ and the actual output $y(t)$. For practicability, the error is only integrated from $t = 0$, when the reference starts to change, to $t = tss$ when the system is in steady-state (after which the margin of control error is within 5%).

The integral operation smooths the fluctuations in the system output due to short-term transients of system states. However, as there are variations due to scheduling and noise,

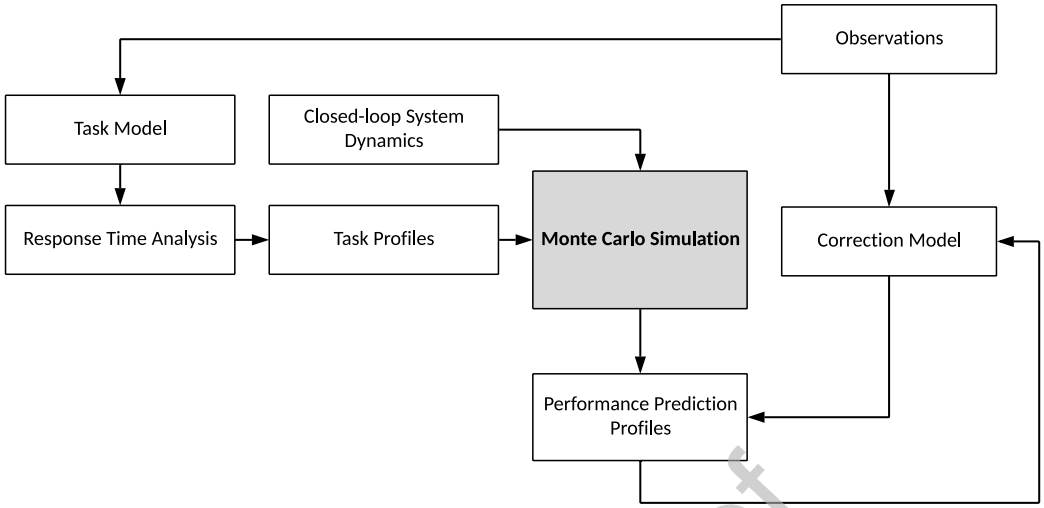


Figure 5: An overview of the Monte Carlo Predictor

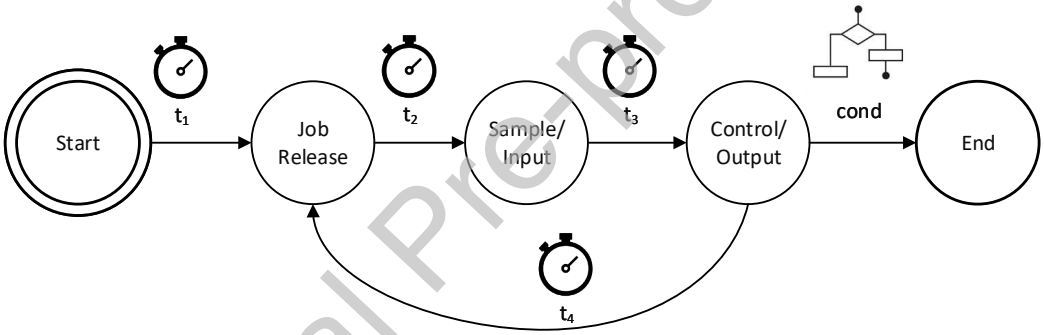


Figure 6: Discrete module that simulates a periodic control task.

each measure of J could be different. In this case, multiple runs of the simulation can be done to obtain a distribution for J. As there is no evidence to prove that the J distribution would follow a certain category of parametric distribution, we use the cumulative distribution function (CDF) to model the data. Depending on the conservativeness of the requirement, the expectation of performance, ~~$\bar{J}$~~ $\hat{J}$, is determined from:

$$\hat{J} = \mathbf{E}[J] = \{X | \mathbf{cdf}(x < X) \geq \alpha_d\} \tag{4}$$

in which $\mathbf{cdf}(\cdot)$ is the cumulative distribution function, and α_d is a decision threshold with $\alpha_d \in (0, 1]$ (with a typical value of 0.95). Note that the definition of expectation in this context is slightly different from the traditional explanation, which is to describe the average output. While in our case, more extreme cases will also need to be considered.

4.3. Prediction Refinement by Error Correction

Ideally, the predictions from the Monte Carlo model are expected to match the measurements of the actual system. However in reality, there are many factors that would affect the

accuracy of the prediction, such as modelling error in the system dynamics, random processes and measurement noises, incorrect assumption of the response time distribution, and integral error due to numerical approximation. This will lead to imprecise and sub-optimal predictions, or even cause the adaptation process to fail.

Many of these errors are impractical to be directly measured or modelled. It is also difficult to analyse how these factors would translate into errors ~~of~~ in performance measurements, even if the error source is identified. As a consequence, a correction model is proposed to refine the predictions in order to handle the errors and improve the utility of the prediction. It is assumed that the predictions ($\hat{J}$) are biased by a factor β with the addition of zero-mean Gaussian noise $\epsilon \in \mathcal{N}(0, \sigma^2)$. Hence the actual performance measurement (J) is given by:

$$J = \hat{J} + \beta + \epsilon \tag{5}$$

In particular, the bias β parameter is estimated with the following criterion:

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & D_n = \sup_x |\mathbf{cdf}_J(x) - \mathbf{cdf}_{\hat{J}}(x)| \\ \text{subject to} \quad & \forall x : \mathbf{cdf}_J(x) > \mathbf{cdf}_{\hat{J}}(x) \end{aligned} \tag{6}$$

in which $\mathbf{cdf}_J(\cdot)$ is the cumulative distribution function of the control cost, $\mathbf{cdf}_{\hat{J}}(\cdot)$ is the cumulative distribution of the estimated control cost, and D_n is the Kolmogorov-Smirnov (K-S) statistic [15] [16], which is the maximal distance between the CDFs of the two distributions. This criterion makes sure the predictions are more conservative than the actual measurements. It is assumed that the prediction error is sustained when making predictions for a small period change, as the main error sources are independent of task period.

5. The Run-Time System

The Monte Carlo simulation introduced in the previous section is executed in the cloud, which significantly reduces the computational load on the local system. In order to achieve adaptation, there is also a need for run-time support on the local computer. The run-time system is formed of two modules: monitor and executor. The monitor module runs on the client side. It is the process for collecting system observations and performing basic conformance analysis. The executor module communicates with the cloud, and is responsible for uploading observed traces, accepting adaptation decisions and informing the kernel to make changes.

The overall flow of the adaptation system is given in Fig. 7. Unlike off-line period assignment methods, the period in this work is updated in multiple iterations. Each iteration only applies a small change to the period. Assuming the control task is running with its initial period T_i^0. When a new request of target utilization is received, a plan is made for changing the current period to the required period, by dividing the objective into fixed small step changes. Based on the system model, a prediction is made by the Monte Carlo Predictor for the new period $T_i' = T_i + \Delta T_i$, in which ΔT_i is the step size. If the predicted performance can satisfy the performance requirement defined by the system, the new period is passed to the scheduler, and the scheduler will change the period of the control task.

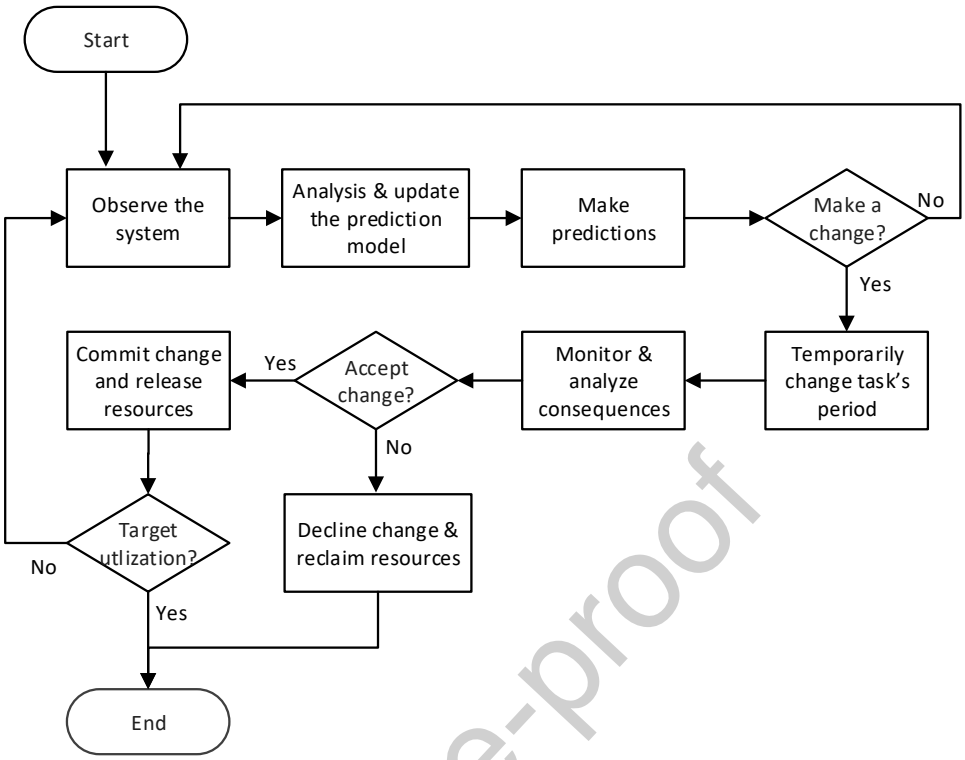


Figure 7: Flowchart of the proposed method (note: this figure is updated)

However, the saved resources will not be immediately available to other tasks. An evaluation phase is then involved to monitor if the system running at the new period can satisfy the performance requirements. If it is satisfied, the saved capacity, $\Delta U = U(T_i) - U'(T_i')$, will be committed and can be used by other tasks in the system. If not, the task's period is returned to its previous value. The prediction model is also updated based on the runtime observations. The process repeats until a) the ~~required~~ targeted utilization is satisfied (which is checked every time after a change is settled); or b) the control performance has reached its bound (i.e. future changes are not sanctioned or are rejected once evaluated on the plant).

6. Evaluation

To demonstrate the effectiveness of the adaptation method, we give an illustrative example that uses a second-order system and a taskset consisting of one control task and multiple non-control tasks. We also evaluate the effectiveness and robustness by investigating a range of design parameters. Unfortunately no currently existing scheme attempts to address the issues identified in this paper, hence a comparative study is not applicable.

The experiment is based on simulation using MATLAB/Simulink. The task scheduler is implemented as a discrete system using the MATLAB s-function in C++, and is called by the

Simulink engine during simulation. The scheduler uses standard fixed-priority scheduling, and the deadline-monotonic policy for task priority assignment. In terms of the controller, we used a Linear-Quadratic-Regulator (LQR) controller. The taskset used in this experiment is randomly generated using UUniFast [17], with log-uniform distributed periods. The schedulability of the taskset is checked through response time analysis.

6.1. Demonstration

We started by evaluating a control application that has one control task and five non-control tasks with higher priorities. ~~Note the tasks with lower priorities will not be affected so they are not considered~~ Note that the tasks with lower priorities than the control task will not have interference thus will not be considered. The system dynamic equation (see Eq.(1)) is defined as follows:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 10 & 25 \\ -25 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix} u, \quad y = \begin{bmatrix} 2.5 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7}$$

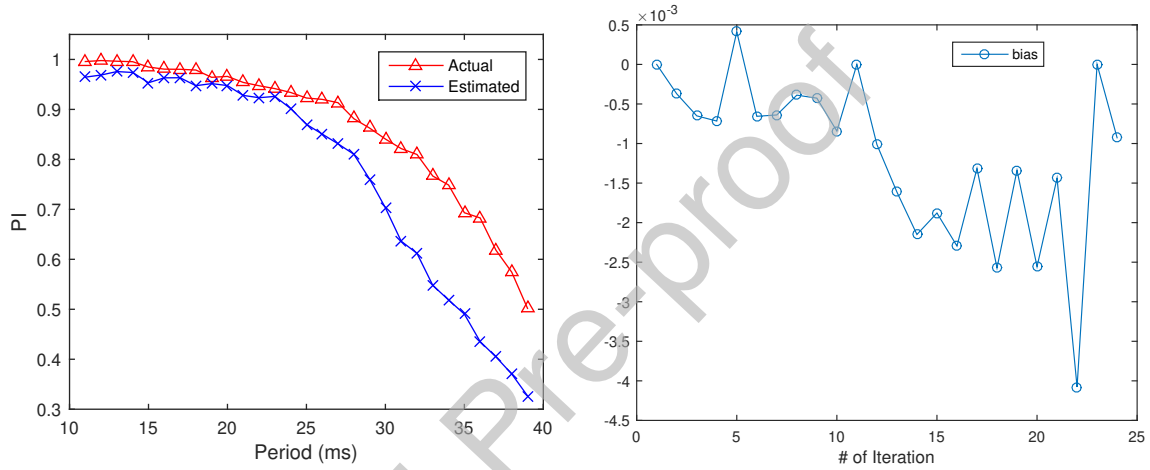
in which the system has a complex conjugate pole pair: $p_{1,2} = 10 \pm 25j$. The closed-loop bandwidth of the system is 40.72 rad/s, which suggests an initial control period of 10 ms (middle rounded value of the usual rule-of-thumb approach). The complete taskset with other higher priority tasks in the system is given in Table 1. At run-time, these tasks are assumed to have variable execution times which are normal distributed from $C_i/2$ to C_i (in this context normal distribution is cut off at 3σ, so it is only defined in the range of $\mu - 3\sigma$ to $\mu + 3\sigma$, with $\sigma = 1/12C_i$ and $\mu = 3/4C_i$). The decision parameter α_d is set to be 0.95, and the period change ΔT_i in each step is 1.0 ms, which is 10% of the initial period.

Table 1: The ~~task set~~ taskset used in the experiment.

Task	C_i (ms)	T_i (ms)	D_i (ms)	Control Task?
τ_0	0.42	1.57	1.57	
τ_1	0.10	2.15	2.15	
τ_2	0.53	4.99	4.99	
τ_3	0.87	7.77	7.77	
τ_4	0.48	8.01	8.01	
τ_5	1.00	10.00	10.00	✓

We first set the degradation factor to 0.7 and run the simulation. For each iteration, the actual system is observed for 1,000 seconds, which will give 400 to 500 measurements depending on the reference signal. The Monte Carlo predictor generates a prediction based on 3,000 randomised task executions and then makes an estimation of the PI for the next step.

The predicted performance is compared with the actual observed metrics in Fig. 8a, and the prediction bias is also shown in Fig. 8b. It can be seen that the predictor made relatively precise predictions, i.e., the deviations between the predictions and observations are small. However, as the period increases, the prediction error also increases. This is explained as the variation of the performance indices increases dramatically when the control period is large. As the predictor is preferred to be more conservative, the extremes that would rarely happen in a real system would still be used to produce expectations. This can ensure the conservativeness of the predictor and reduce the chance of invalidation. For this experimental run, the period is terminated by the predictor at 39 ms, which is four times the initial period, i.e., the utilization is only 25% of the initial task utilization.

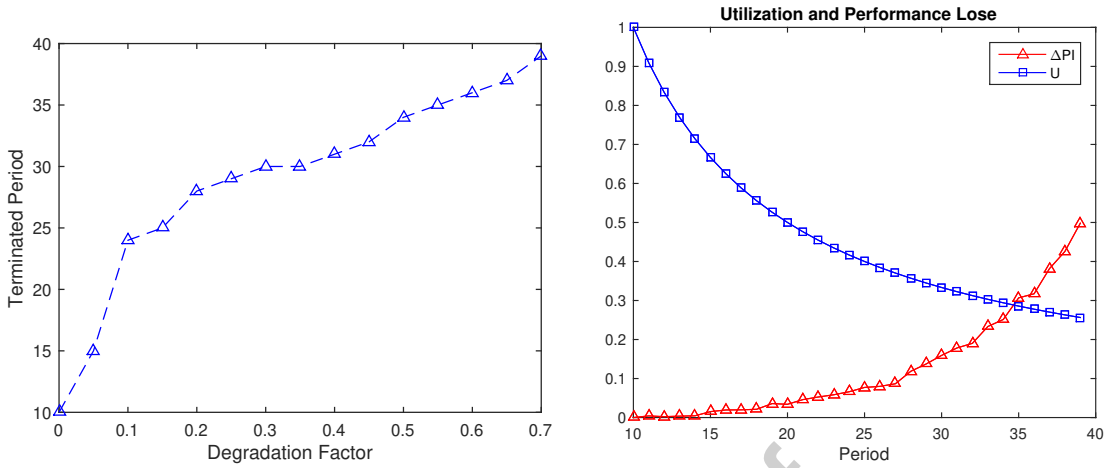


(a) Predicted v.s. Actual PIs (b) Prediction bias estimation

Figure 8: Experiment results. Each point in the diagrams represents one iteration.

For an actual system, a degradation of 0.7 may be impractical as the control performance drop could be too high. To give a full spectrum of the system behaviour, we used a range of degradation factors from 0.05 to 0.70. From Fig. 9a, we can see as the degradation factor increases, the period that the algorithm terminates at also dramatically increases. For example, if $\lambda_d = 0.1$, the period can be 25 ms, while if $\lambda_d = 0.5$, the period can be 34 ms. It can be seen the degradation factor is an important design parameter as it determines when the adaptation process will have to be terminated.

To better illustrate the trade-off between utilization and performance, we compare the two metrics against task period in Fig. 9b. It can be seen that as the period increases, the control performance loss is also gradually increased. On the other hand, task utilization is reduced as a consequence of using a longer period. However, the benefit of utilization saved by increasing task period is exponentially decreased, while the penalty to control performance grows quadratically. This implies that increasing the control task's period could have a significant positive impact on scheduling performance in terms of utilization, with just a small amount of control degradation as penalty. However, over-extending the period of a control task could dramatically affect control performance while making limited



(a) Terminated period v.s. λ_d (b) Utilisation v.s. PI

Figure 9: Exploring the Degradation Factor

contribution to utilization saving.

This trade-off is controlled by the degradation factor. The degradation factor is an important design parameter in guiding the adaptation process. As it is a relative measure and it is used mainly for facilitating numerical computation, sometimes it is not straightforward to select a proper parameter. In this experiment, it is suggested that the degradation factor would be less than 0.3 to obtain a good trade-off. However, we used an unstable system in the experiment which is more sensitive to period changes. For other systems, the degradation factor would be higher. The selection of a proper degradation factor could be done by off-line simulation with conservative conditions.

6.2. Robustness

In the previous demonstration, we assumed the actual system is identical to the design model. In this experiment, we will explore the robustness of the algorithm by looking at the case in which model mismatch exists (i.e., the system model deviates from the actual system). ~~Model mismatch is the phenomenon that the system model is deviated from the actual system.~~ This is common in actual engineering systems, due to many factors such as modelling error, limited knowledge of the system, and simplification of the physical system.

We evaluate the robustness by applying deviations into the actual system, compared to the design model. As model differences exist, the actual system behaviour will be different from expected. This includes both the system dynamic model and the task model. The experiment configurations are given in Table 2. There are overall 7 experiments including a baseline E_1 (the one described above). The 'system dynamics' column in the table indicates how much percentage of error is added to the system dynamic matrix A_i (defined in Eq.(1)). The 'task model' column indicates what task model is used in the simulation: 'WCET' for always using the worst-case execution times, 'BCET' for the best-case execution times, and 'NORM' for normally distributed between these two extremes.

Table 2: Experiment configuration for evaluating robustness

Experiment	System dynamics error	Task execution model
E_1	0%	NORM
E_2	-5%	
E_3	+5%	
E_4	0%	WCET
E_5	+5%	
E_6	0%	BCET
E_7	-5%	

The results are shown in Fig. 10. From the figure it can be seen that the actual system outputs could be very different if there are errors in the original model. The worst-case is E_5 in which both system dynamics and task model are worse than ideal. The best two cases are E_6 and E_7 in which the best-case execution time model is applied. Nevertheless, in all cases, the predictions are more conservative than the actual observations, and it can be seen the predictor is able to correct itself to accommodate different situations. For example, in E_6 and E_7, as the actual observations are much better than the model, the prediction corrects itself so it will not be too conservative.

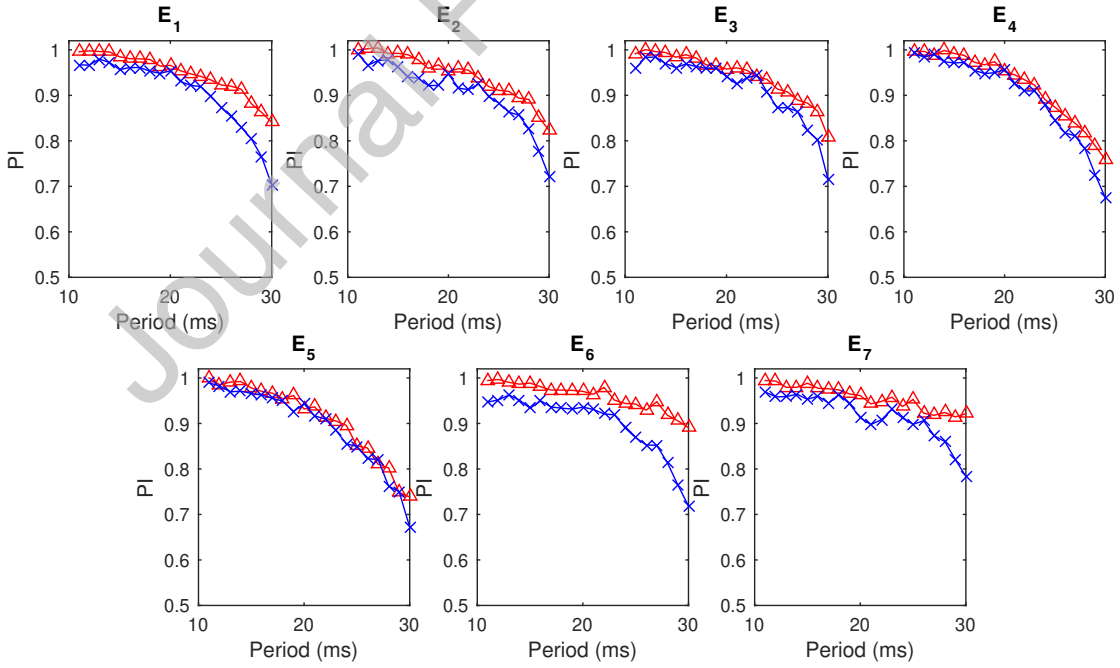


Figure 10: Results of robustness evaluation. The lines with 'x' marks are the predictions and the lines with '△' marks are the actual observations.

7. Analysis and Discussion

From the standard response time analysis, we can see that extending the period of a task will not change the worst-case response time of that task. Also, it will have no influence on higher priority tasks. Thus the schedulability of the system is sustained [18, 19]. The priority ordering will be unaffected if the priority assignment policy is deadline-monotonic as task deadlines are not changed. However, if rate-monotonic priority ordering is used (and deadlines change with period), extending the period without changing priorities would affect the optimality of the priority ordering.

7.1. Implementation Overhead

Although most of the computation load is distributed to the cloud server, the implementation of the method still requires additional computation and communication in the local embedded computer. In particular, these are:

Computation overhead: the additional overhead of computation mainly comes from calculating performance statistics and run-time monitoring. The process can be run as a low priority background service to create minimum interferences to other running tasks in the system.

Memory overhead: as traces and statistical data is buffered into memory before being sent to the cloud server, some memory storage is required. Depending on the sampling rate and reliability of the communication link, this size can range from a hundred bytes to a few kilobytes.

Communication overhead: the communication overhead is minor. As only packets containing statistical data are transferred to the cloud server, the communication bandwidth required by the method is negligible. Also as the communication is not in the control loop, the real-time and reliability requirements of the network are also low.

Cloud computing cost: as data is collected and analysed on the cloud, its cost needs to be considered. Most of the cloud service provides a variety of configurations, including number of cores, memory size and communication bandwidth. For our method, the requirement of performance is low, hence even a minimal configuration (e.g., 1-core CPU @ 2.0 GHz, 2 GB RAM, 200 MB disk) with relatively low cost could work. Considering that most CPS now have cloud services already, the expected cost would be even lower.

7.2. Multiple Control Tasks

In the experimental evaluation we considered the case in which only one control task exists in the system. For a more general case in which multiple control tasks exist in the system, it is essential to prioritise each task for adaptation. To control the influence of adaptation, we recommend that only one task at a time is in an adaptation cycle and can change its period. Assuming all control tasks are equally important, we propose the following three policies that can prioritise control tasks:

Highest Priority First (HPF) The higher priority task has the largest margin and the highest interferences with non-control tasks, and also changing the higher priority task first can avoid the need for recalculating periods of lower priority task.

Least Sensitivity First (LSF) The task is selected according to the sensitivity of its performance index by changing its period. This is evaluated by a sensitivity function:

$$\Delta J = \frac{\partial J}{\partial h} \Delta h \Big|_{h=T_i(k-1)} \tag{8}$$

Least Uncertainty First (LUF) Select the controller that behaves closest to its prediction model, i.e., minimal prediction bias and error.

If the importance, or critical level, of the control tasks are not equal, it is always preferable to change the task with the least importance first.

7.3. Implementation in Ada

It is important to consider, when evaluating a new software architecture, whether the necessary facilities and primitives are actually available in the tools and languages used to implement current embedded and cyber-physical systems. In this section we briefly assess the capabilities of Ada in terms of the support it can provide for the period adaption scheme introduced in this paper. However, before considering these specific requirements it is perhaps useful to highlight the main features that the Ada programming language support that aid in the production of real-time software:

- support for concurrent programming, both static and dynamic models, and including input and output jitter control;
- access to clocks, delay primitives and timeout recognition;
- low-level programming facilities that allows device drivers and interrupt handlers to be programmed;
- support for fixed priority and EDF (Earliest Deadline First) scheduling – including direct support for priority ceiling and deadline floor protocols;
- support for execution time budget control – individual tasks, or groups of tasks, can execute within defined execution-time budgets, tasks can be programmed to be suspended if they run out of budget;
- support for exception handling and other language features that aid the production of fault-free and fault tolerant software.

In addition to these facilities the period adaption scheme requires some additional capabilities:

1. Monitoring task execution times — each task in Ada has an execution time clock that is updated automatically by the run-time support system (run-time kernel) and which can be read by any other task; a low priority monitoring task can therefore obtain the current clock values from all control tasks.

2. Modifying task execution rates — a periodic task is implemented, in Ada, as a repeating task within which is a delay statement enforces a cycle time (e.g. $25ms$); this parameter in under the control of the program and hence a planner task can make changes to this value in any control task (e.g. increase it to $26ms$).

3. Modifying task priorities — if task periods and deadlines are changed sufficiently that the initial priority assignment is no longer optimal then task priorities can be updated using Ada's predefined 'Dynamic_Priorities' package; priority ceiling levels can also be modified to keep then consistent with the tasks' priorities.

4. Modifying task execution time budgets — in terms of the adaptability of the complete system, as execution time is made available from control tasks executing less frequently, it can be assigned to the budgets of non-control tasks that are able to utilise this new capacity - Ada allows dynamic budget management to be programmed.

5. Modifying processor speed — although Ada does not provide any facilities for directly controlling the speed of the processor on which the program is executing, it does allow code to directly manipulate processor registers; hence if the processor does allow speed (and hence voltage) to be changed then this capability can be managed at the program level.

In summary, Ada provides all the necessary features needed to fully implement to proposed scheme.

8. Related Work (this is a new section)

This work contributes to the research on control-scheduling co-design [13, 20, 21]. In the co-design research, task scheduling is considered explicitly and simultaneously with the design process for the controllers. The novelty of our work is the idea of digital twining and utilising historic information to support run-time changes. Also the resource constraint on the control task varies as a consequence of evolutionary changes from other parts of the system.

The decision process forms a loop similar to those in feedback scheduling [22, 23]. For feedback scheduling, the scheduler has the ability to monitor system states and make corresponding actions by adjusting scheduling parameters, including task periods, execution times, deadlines and priorities. In this work, we focus on changing task periods and we have introduced cloud services into the feedback loop to support more complicated decisions.

The work of optimal control period assignment [12, 11] provides a solution for optimally select the control period based some performance expectations. In contrast to these method

that are performed off-line, our work is a run-time method that focus on evolutionary changes that are not possible to model at design-time. This also distinguish our work from graceful degradation [24], in which the degradation is planned and pre-designed based on known possible failures.

Finally, there is an active research on state-aware and resource-aware feedback period allocation methods [25, 26, 27]. The idea of this work is to change task period frequently based on system states that are measured every tens or hundreds of milliseconds. The philosophy we applied is different: the control task periods is gradually and slowly changed, in which a change could be made across a large time span, e.g, hours or days. This makes our approach much less dynamic than other feedback-based period allocation methods, but our scheme focuses on making permanent changes to task periods. The changes our scheme sanctions are also safe due to the confidence obtain from observing the consequences of any change. We also have less run-time overhead on the target processor due to the use of cloud-based computing.

9. Conclusions

In this work, we proposed an adaptive scheduling framework that can accommodate additional computing requirements (or reduce power consumption) for cyber-physical systems at run-time. A cloud facility is proposed as a component in the loop ~~of~~ for monitoring and improving control and scheduling performance. Also, we designed a method that uses a system dynamics model and a task timing model to make predictions and instigate future actions. Finally we contributed a scheme which utilises run-time feedback information to enhance the precision and robustness of the predictions. The proposed method is demonstrated through experiments using simulations with one specific second-order system and a random generated taskset (a more comprehensive evaluation forms a further work). ~~A few discussions~~ Discussions on the implementation including overhead, support of multiple control tasks and implementation in Ada are also given.

There are still many other aspects we can further explore. These include consideration of the case in which the system has dependent control tasks, for example a multi-loop controller that has cascaded control tasks. Another example is a motion control system with multiple degrees of freedom, e.g., a humanoid robot, in which more than one control task has to cooperate and be synchronised. We also want to extend our method to support adaptation with multiple objectives, in which more than one design objectives and even conflicted constraints need to be considered. The ultimate aim is to integrate this work with detection and prediction methods, so the changes in the system can be reflected and seamlessly compensated by the adaptation.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

References

[1] X. Dai, A. Burns, Predicting worst-case execution time trends in long-lived real-time systems, in: Ada-Europe International Conference on Reliable Software Technologies, Springer, 2017, pp. 87–101.

[2] X. Dai, Flexible and adaptive real-time task scheduling in cyber-physical control systems, Ph.D. thesis, University of York (2018).

[3] F. Kerasiotis, A. Prayati, C. Antonopoulos, C. Koulamas, G. Papadopoulos, Battery lifetime prediction model for a wsn platform, in: Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on, IEEE, 2010, pp. 525–530.

[4] B. Brock, K. Rajamani, Dynamic power management for embedded systems, in: IEEE International Systems-on-Chip Conference, 2003. Proceedings., IEEE, 2003, pp. 416–419.

[5] J. W. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, W. Zhao, Algorithms for scheduling imprecise computations, Springer, 1991.

[6] K.-J. Lin, S. Natarajan, Expressing and maintaining timing constraints in flex, in: Real-Time Systems Symposium, 1988., Proceedings., IEEE, 1988, pp. 96–105.

[7] D. Hull, W.-c. Feng, J. W. Liu, Operating system support for imprecise computation, Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities.

[8] K.-E. Årzén, A. Cervin, D. Henriksson, Implementation-aware embedded control systems, Handbook of networked and embedded control systems (2005) 377–394.

[9] K.-E. Årzén, A. Cervin, Software and platform issues in feedback control systems, Cyber-Physical Systems (2017) 165–195.

[10] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzén, G. Buttazzo, The jitter margin and its application in the design of real-time control systems, in: Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications, Gothenburg, Sweden, 2004, pp. 1–9.

[11] M. Ryu, S. Hong, Toward automatic synthesis of schedulable real-time controllers, Integrated Computer-Aided Engineering 5 (3) (1998) 261–277.

[12] D. Seto, J. P. Lehoczky, L. Sha, K. G. Shin, On task schedulability in real-time control systems, in: Real-Time Systems Symposium, 1996., 17th IEEE, IEEE, 1996, pp. 13–21.

[13] K.-E. Arzén, A. Cervin, J. Eker, L. Sha, An introduction to control and scheduling co-design, in: Decision and Control, 2000. Proceedings of the 39th IEEE Conference on, Vol. 5, IEEE, 2000, pp. 4865–4870.

[14] T. Chantem, X. Wang, M. D. Lemmon, X. S. Hu, Period and deadline selection for schedulability in real-time systems, in: Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on, IEEE, 2008, pp. 168–177.

[15] J. S. Cho, M.-H. Park, P. C. Phillips, Practical kolmogorov–smirnov testing by minimum distance applied to measure top income shares in korea, Journal of Business & Economic Statistics (2017) 1–15.

[16] C. Whitnall, E. Oswald, L. Mather, An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis, in: International Conference on Smart Card Research and Advanced Applications, Springer, 2011, pp. 234–251.

[17] E. Bini, G. C. Buttazzo, Measuring the performance of schedulability tests, Real-Time Systems 30 (1-2) (2005) 129–154.

[18] S. Baruah, A. Burns, Sustainable schedulability analysis, in: Proc. of IEEE Real-Time Systems Symposium (RTSS), 2006, pp. 159–168.

[19] A. Burns, S. Baruah, Sustainability in real-time scheduling, Journal of Computing Science and Engineering 2 (1) (2008) 74–97.

[20] K.-E. Årzén, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, L. Sha, Integrated control and scheduling, Department of Automatic Control, Lund Institute of Technology, Sweden, Tech. Rep. ISRN LUTFD2/TFRT–7586–SE.

[21] X. Dai, W. Chang, S. Zhao, A. Burns, A dual-mode strategy for performance-maximisation and resource-efficient cps design, ACM Transactions on Embedded Computing Systems (TECS) 18 (5s) (2019) 85.

[22] C. Lu, J. Stankovic, G. Tao, S. H. Son, et al., Design and evaluation of a feedback control edf scheduling

algorithm, in: Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE, IEEE, 1999, pp. 56–67.

[23] C. Lu, J. A. Stankovic, S. H. Son, G. Tao, Feedback control real-time scheduling: Framework, modeling, and algorithms, Real-Time Systems 23 (1-2) (2002) 85–126.

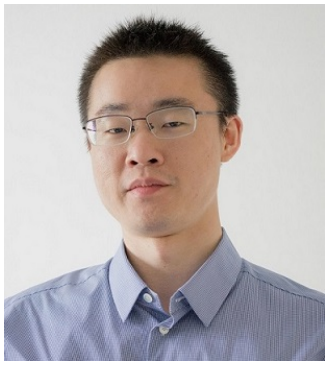
[24] K. G. Shin, C. L. Meissner, Adaptation and graceful degradation of control system performance by task reallocation and period adjustment, in: Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on, IEEE, 1999, pp. 29–36.

[25] M. M. B. Gaid, A. Cela, Y. Hamam, C. Ionete, Optimal scheduling of control tasks with state feedback resource allocation, in: American Control Conference, 2006, IEEE, 2006, pp. 6–pp.

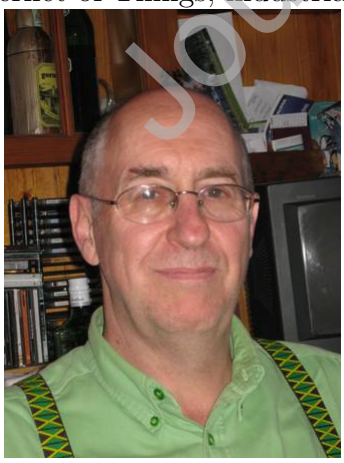
[26] P. Marti, C. Lin, S. A. Brandt, M. Velasco, J. M. Fuertes, Optimal state feedback based resource allocation for resource-constrained control tasks, in: Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International, IEEE, 2004, pp. 161–172.

[27] C. Lozoya, P. Martí, M. Velasco, J. M. Fuertes, Control performance evaluation of selected methods of feedback scheduling of real-time control tasks, IFAC Proceedings Volumes 41 (2) (2008) 10668–10673.

Biography



Xiaotian Dai is a research associate of the Department of Computer Science at University of York, UK. He has research interests in real-time systems and systems engineering, including task scheduling, control scheduling co-design, flexible and adaptive scheduling and system safety assurance. He has worked on a wide range of cyber-physical systems, e.g., Internet-of-Things, industrial control and autonomous systems.



Alan Burns: Professor Alan Burns is a member of the Department of Computer Science, University of York, U.K. His research interests cover a number of aspects of real-time systems

including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to real-time applications. His teaching activities include courses in Operating Systems and Real-time Systems. In 2009 Professor Burns was elected a Fellow of the Royal Academy of Engineering. In 2012 he was elected a Fellow of the IEEE.